What is claimed:

1. A method of translating compiled programming code from a first code state to a second code state, the programming code in the first code state comprising a plurality of basic blocks, each basic block comprising a set of instructions, at least one basic block ending in a dynamic branch, the dynamic branch being a transfer to one of a set of destinations based on a calculation of a destination address, the method comprising the steps of:

identifying the plurality of basic blocks in the first code state of the programming code;

identifying links between the identified basic blocks;

constructing a control flow graph / representation (CFG) of the programming code based on the identified basic blocks and identified links, the CFG being in a preliminary form;

identifying at least one basic block ending in a dynamic branch;

exploring, based on the CFG, all identified basic blocks that lead to the dynamic branch as far back as is necessary to fully determine a set of destination addresses for the dynamic branch, the set of destination addresses defining the set of destinations from the dynamic branch;

examining the set of destinations to identify a branch table;

updating the CFG to reflect the set of destinations and the identified branch table;

translating the programming code from the first code state to the second code state based at least in part on the updated CFG.

2. The method of claim 1 wherein the exploring step comprises the steps of:

for each explored basic block, constructing a corresponding code graph / representation (code graph) of the instructions in such basic block; and

traversing each code graph to determine the set of destination addresses from the dynamic branch.

3. The method of claim 2 wherein each code graph is a rooted directed acyclic graph having interconnected nodes, each node being one of:

an instruction node representing an instruction in the corresponding basic block;

an argument node representing an argument in the corresponding basic block;

an apply node edging to an instruction node and to an argument node and representing the application of such argument node to such instruction node, the apply node in certain instances also being an argument node edged to by another node;

a stack node edging to a pair of argument nodes and acting as an argument node having the pair of argument nodes;

a missing argument node representing a missing argument supplied from a different basic block; and

an alias node edged to by a stack node or apply node and edging to an argument remote from such stack node, and representing such remote argument to such stack node.

4. The method of claim 3 wherein the exploring step comprises the steps of:

noting that a first code graph corresponding to a first basic block contains a missing / insufficiently defined argument node;

locating, based on the CFG, a second basic block that immediately precedes the first basic block, a second code graph corresponding to the second basic block;

constructing a new code graph having the first code graph and the second code graph by replacing the missing argument node in the first code graph with the second code graph; and

traversing the new code graph in an effort to determine the set of destination addresses from the dynamic branch.

5. The method of claim 4 wherein the exploring step comprises the steps of:

locating, in the portion of the new code graph corresponding to the second code graph, an instruction node representing a condition that must be satisfied to allow program flow to continue from the second basic block to the first basic block; and

replacing the located instruction node with a test node indicative of the condition that must be satisfied.

6. The method of claim 5 wherein the located instruction node is representative of a branch instruction, and the condition that must be satisfied is a logical true, a logical false, or a pre-determined index value.

7. The method of claim 5 wherein the exploring step comprises the step of performing bounds evaluation according to a list of props, each prop indicating a target node to which such prop applies and a set of bounds that are to be applied to such node.

8. The method of claim 7 wherein the step of performing bounds evaluation comprises the step of traversing each node in the code graph for each prop in the list.

9. The method of claim 8 wherein the step of traversing each node comprises, for each prop, the step of attempting to push bounds information into the argument node edged to from an apply node if such apply node is being traversed and is the target node of the prop.

10. The method of claim 9 wherein each node includes bounds information, and wherein the attempting step comprises the steps of:

attempting to deduce bounds information for the argument node edged to from the target apply node based on such argument node and the instruction node edged to from such target apply node;

if appropriate, altering the bounds information of the argument node edged to from the target apply node; and

adding a new prop to the list if the bounds information of the argument node edged to from the target apply node is altered.

11. The method of claim 7 wherein the step of traversing each node comprises creating a prop for the argument of each test node the first time the test node is encountered.

12. The method of claim 11 wherein the step of traversing each node comprises preliminarily traversing the code graph with an empty prop the purpose of accumulating props for test nodes.

13. The method of claim 12 wherein each node includes bounds information, and wherein the step of traversing each node comprises stepping through the code graph in a spanning direction, and then re-tracing through the code graph opposite the spanning direction, the re-tracing step including the step of re-evaluating the bounds information of each re-traced node whenever a change in bounds may result.

14. A translator operating on a processor for translating compiled programming code from a first code state to a second code state, the programming code in the first code state comprising a plurality of basic blocks, each basic block comprising a set of instructions, at least one basic block ending in a dynamic branch, the dynamic branch being a transfer to one of a set of destinations based on a calculation of a destination address, the translator:

identifying the plurality of basic blocks in the first code state of the programming code;

identifying links between the identified basic blocks;

constructing a control flow graph / representation (CFG) of the programming code based on the identified basic blocks and identified links, the CFG being in a preliminary form;

identifying at least one basic block ending in a dynamic branch;

exploring, based on the CFG, all identified basic blocks that lead to the dynamic branch as far back as is necessary to fully determine a set of destination addresses for the dynamic branch, the set of destination addresses defining the set of destinations from the dynamic branch;

examining the set of destinations to identify a branch table;

updating the CFG to reflect the set of destinations and the identified branch table;

translating the programming code from the first code state to the second code state based at least in part on the updated CFG.

15. The translator of claim 14 wherein the translator:

for each explored basic block, constructs a corresponding code graph / representation (code graph) of the instructions in such basic block; and

traverses each code graph to determine the set of destination addresses from the dynamic branch.

16. The translator of claim 15 wherein each code graph is a rooted directed acyclic graph having interconnected nodes, each node being one of:

an instruction node representing an instruction in the corresponding basic block;

an argument node representing an argument in the corresponding basic block;

an apply node edging to an instruction node and to an argument node and representing the application of such argument node to such instruction node, the apply node in certain instances also being an argument node edged to by another node;

a stack node edging to a pair of argument nodes and acting as an argument node having the pair of argument nodes;

a missing argument node representing a missing argument supplied from a different basic block; and an alias node edged to by a stack node or apply node and edging to an argument remote from such stack node, and representing such remote argument to such stack node.

17. The translator of claim 16 wherein the translator:

notes that a first code graph corresponding to a first basic block contains a missing / insufficiently defined argument node;

locates, based on the CFG, a second basic block that immediately precedes the first basic block, a second code graph corresponding to the second basic block;

constructs a new code graph having the first code graph and the second code graph by replacing the missing argument node in the first code graph with the second code graph; and

traverses the new code graph in an effort to determine the set of destination addresses from the dynamic branch.

18. The translator of claim 17 wherein the translator:

locates, in the portion of the new code graph corresponding to the second code graph, an instruction node representing a condition that must be satisfied to allow program flow to continue from the second basic block to the first basic block; and

replaces the located instruction node with a test node indicative of the condition that must be satisfied.

19. The translator of claim 18 wherein the located instruction node is representative of a branch instruction, and the condition that must be satisfied is a logical true, a logical false, or a pre-determined index value.

20. The translator of claim 18 wherein the translator performs bounds evaluation according to a list of props, each prop indicating a target node to which such prop applies and a set of bounds that are to be applied to such node.

21. The translator of claim 20 wherein the translator traverses each node in the code graph for each prop in the list.

22. The translator of claim 21 wherein the translator, for each prop, attempts to push bounds information into the argument node edged to from an apply node if such apply node is being traversed and is the target node of the prop.

23. The translator of claim 22 wherein each node includes bounds information, and wherein the translator:

attempts to deduce bounds information for the argument node edged to from the target apply node based on such argument node and the instruction node edged to from such target apply node;

if appropriate, alters the bounds information of the argument node edged to from the target apply node; and

adds a new prop to the list if the bounds information of the argument node edged to from the target apply node is altered.

24. The translator of claim 20 wherein the translator creates a prop for the argument of each test node the first time the test node is encountered.

25. The translator of claim 24 wherein the translator preliminarily traverses the code graph with an empty prop the purpose of accumulating props for test nodes.

26. The translator of claim 25 wherein each node includes bounds information, and wherein the translator steps through the code graph in a spanning direction, and then re-traces through the code graph opposite the spanning direction, the re-tracing including re-evaluation of the bounds information of each re-traced node whenever a change in bounds may result.

27. In connection with translating compiled programming code from a

first code state to a second code state, the programming code in the first code state comprising a plurality of basic blocks, each basic block comprising a set of instructions, at least one basic block ending in a dynamic branch, the dynamic branch being a transfer to one of a set of destinations based on a calculation of a destination address, a computer-readable medium having computer-executable instructions for performing steps comprising:

identifying the plurality of basic blocks in the first code state of the programming code;

identifying links between the identified basic blocks;

constructing a control flow graph / representation (CFG) of the programming code based on the identified basic blocks and identified links, the CFG being in a preliminary form;

identifying at least one basic block ending in a dynamic branch;

exploring, based on the CFG, all identified basic blocks that lead to the dynamic branch as far back as is necessary to fully determine a set of destination addresses for the dynamic branch, the set of destination addresses defining the set of destinations from the dynamic branch;

examining the set of destinations to identify a branch table;

updating the CFG to reflect the set of destinations and the identified branch table;

translating the programming code from the first code state to the second code state based at least in part on the updated CFG.

28. The computer-readable medium of claim 27 wherein the exploring step comprises the steps of:

for each explored basic block, constructing a corresponding code graph / representation (code graph) of the instructions in such basic block; and

traversing each code graph to determine the set of destination addresses from the dynamic branch.

29. The computer-readable medium of claim 28 wherein each code

graph is a rooted directed acyclic graph having interconnected nodes, each node being one of:

an instruction node representing an instruction in the corresponding basic block;

an argument node representing an argument in the corresponding basic block;

an apply node edging to an instruction node and to an argument node and representing the application of such argument node to such instruction node, the apply node in certain instances also being an argument node edged to by another node;

a stack node edging to a pair of argument nodes and acting as an argument node having the pair of argument nodes;

a missing argument node representing a missing argument supplied from a different basic block; and

an alias node edged to by a stack node or apply node and edging to an argument remote from such stack node, and representing such remote argument to such stack node.

30. The computer-readable medium of claim 29 wherein the exploring step comprises the steps of:

noting that a first code graph corresponding to a first basic block contains a missing / insufficiently defined argument node;

locating, based on the CFG, a second basic block that immediately precedes the first basic block, a second code graph corresponding to the second basic block;

constructing a new code graph having the first code graph and the second code graph by replacing the missing argument node in the first code graph with the second code graph; and

traversing the new code graph in an effort to determine the set of destination addresses from the dynamic branch.

31. The computer-readable medium of claim 30 wherein the exploring

step comprises the steps of:

locating, in the portion of the new code graph corresponding to the second code graph, an instruction node representing a condition that must be satisfied to allow program flow to continue from the second basic block to the first basic block; and

replacing the located instruction node with a test node indicative of the condition that must be satisfied.

32. The computer-readable medium of claim 31 wherein the located instruction node is representative of a branch instruction, and the condition that must be satisfied is a logical true, a logical false, or a pre-determined index value.

33. The computer-readable medium of claim 31 wherein the exploring step comprises the step of performing bounds evaluation according to a list of props, each prop indicating a target node to which such prop applies and a set of bounds that are to be applied to such node.

34. The computer-readable medium of claim 33 wherein the step of performing bounds evaluation comprises the step of traversing each node in the code graph for each prop in the list.

35. The computer-readable medium of claim 34 wherein the step of traversing each node comprises, for each prop, the step of attempting to push bounds information into the argument node edged to from an apply node if such apply node is being traversed and is the target node of the prop.

36. The computer-readable medium of claim 35 wherein each node includes bounds information, and wherein the attempting step comprises the steps of:

attempting to deduce bounds information for the argument node edged to from the target apply node based on such argument node and the instruction node edged to from such target apply node;

if appropriate, altering the bounds information of the argument node edged to from the target apply node; and

adding a new prop to the list if the bounds information of the argument node edged to from the target apply node is altered.

37. The computer-readable medium of claim 33 wherein the step of traversing each node comprises creating a prop for the argument of each test node the first time the test node is encountered.

38. The computer-readable medium of claim 37 wherein the step of traversing each node comprises preliminarily traversing the code graph with an empty prop the purpose of accumulating props for test nodes.

39. The computer-readable medium of claim 38 wherein each node includes bounds information, and wherein the step of traversing each node comprises stepping through the code graph in a spanning direction, and then re-tracing through the code graph opposite the spanning direction, the re-tracing step including the step of re-evaluating the bounds information of each re-traced node whenever a change in bounds may result.